Proceedings of the 5th WSEAS International Conference on Applications of Electrical Engineering, Prague, Czech Republic, March 12-14, 2006 (pp147-151)

Exploiting Cryptographic Architectures over Hardware Vs. Software Implementations: Advantages and Trade-Offs

N. SKLAVOS [I], K. TOULIOU [II], and C. EFSTATHIOU [III]

[I] Electrical and Computer Engineering Dept., University of Patras, GREECE
[II] Department COMELEC, Ecole National Supérieure des Télécommunications
46, rue Barrault, 75634 Paris Cedex 13, FRANCE
[III] Computer Dept., Technological Educational Institute, ATEI of Athens, GREECE

Abstract: - Cryptographic modules can be implemented in both hardware and software. Although software cryptographic implementations are cost-effective and more flexible, they seem to provide a much lower level of security in relation to their hardware equivalents. The uncontrolled memory access, the vulnerabilities imposed by the OS and the facility of modifying software implementations are some of the security barriers of software cryptographic modules. This works deals with the exploitation of security architectures via software and hardware implementations. Especially it is centers in the advantages and the trade-offs of each one of the two alternatives integration approaches.

Key-Words: - Software/hardware cryptographic implementations, memory access, reverse-engineering, timing and power analysis attacks, OS vulnerabilities, random number generators, key storage.

1 Introduction

When dealing with the implementation of cryptographic modules, one of the questions raised is whether to deploy software products or hardware based solutions. Software security implementations are designed using programming languages such as C, C++, Java or even Assembly with the intention to run on general-purpose microprocessors, digital signal processors or to be embedded in smart cards. On the other hand, hardware cryptographic modules are designed with hardware description languages such as VHDL or Verilog. They are mainly implemented in *Field Programmable Gate Arrays* (FPGAs) and *Application Specific Integrated Circuits* (ASICs) [8].

The debate over hardware vs. software cryptographic designed architectures and also implementations seems endless [19]. Each one of the two approaches has its advantages and trade-offs. The performance specifications of the application, the desirable cost and the security demands are the main elements that determine each time the best and most applicable solution.

However, aspects such as performance, cost or flexibility are beyond the scope of this article. Our interest is focused on the comparison between hardware and software based cryptographic implementations only in terms of the security they provide.

This paper is organized as follows: In *Section 1* a detailed introduction to the focussed topic is given. In *Section 2* Software Security Limitations are given in detail. In the next *Section 3*, the combination of both hardware and software integration platforms are presented. Finally, conclusions and outlook are given in *Section 4*.

2 Software Security Limitations

Although software security implementations are cost-effective, more flexible and easy for development and possible upgrade, they seem to provide a much lower level of security in relation to their hardware equivalents [1], [4], [5]. The software modules weakness to provide sufficient protection lies on the following reasons:

- *Security bound imposed by the OS*
- *Arbitrary Memory Access*
- *Lack of Data Integrity Guarantee*
- *Insecure Key Storage*
- *Pseudo-Random Number Generators*
- *No resistance against reverse-engineering*
- *Vulnerability to Side Channel Attacks*

Each one of these software security barriers is further analyzed in the following sections.

2.1 Security bound imposed by the OS

Software cryptographic modules, when executed on general-purpose computers, do not exist in isolation. They run always on top of another lower layer application [1], [5], such as an operating system

(OS), which inevitably imposes an upper bound on the level of the security efficiency [1]. No matter how well secure the software module is, if there is a bug or information leakage on the OS, the whole implementation is risking a possible attack. In other words, the protection provided by the software-based implementation depends on the OS security level.

Unfortunately, this dependence on the OS leaves a lot of doubts about whether or not we can trust software cryptographic modules. Knowing, also that today's OS suffer from information flaws which make them susceptible to various attacks, software cryptographic modules do not appear to be the best candidate in terms of security. Some of the common OS problems are listed below:

- *Vulnerabilities (Viruses, Trojan horses) [1], [13], [18]*
- *Memory Management malfunction (Buffer overflows) [3], [11]*
- *Memory contents leakage (Swap Files) [1], [3], [18]*

On the other hand, hardware cryptographic modules are implemented below the operating systems and thus, they are not threatened by attacks against the latter or by flaws of another higher layer application [1], [5].

2.2 Arbitrary Memory Access

One of the major disadvantages of the software cryptographic implementations is that they do not employ their own physical memory. Usually, they use an external memory, controlled by the operating system they are running on [1]. However, this memory space is shared between other applications that might be executed at the same time. Allowing to other processes an access to the common memory, there is no assurance that the contents of the memory space, used by the cryptographic module, will not be read or even altered by an unauthorized application [1], [15].

For example, Windows NT provide a function called *ReadProcessMemory()*, which allows a process to read the memory of almost any other process in the system [16]. The arbitrary memory access is not only a Windows' drawback since in many Unix systems the use of *ptrace* imposes similar risks to those arising from the *ReadProcessMemory()* function [16]. Although all operation systems provide some sort of memory access protection, its efficiency depends on the robustness of the system and its vulnerability to information flaws.

In order to realize how important is the threat arising from the uncontrolled memory access, we should consider that during the execution of a cryptographic algorithm, internal values of the key need to be stored temporarily in the memory. Moreover, since the efficiency of the cryptographic module is determined by how well protected the values of the cipher key are, and if no guarantee can be made concerning their security then the corresponding implementation is untrustworthy.

The superiority of the hardware cryptographic modules compared to the software ones is based on the formers' privilege to facilitate their own physical memory [1], [10], [15]. Having the memory space completely in their disposal, hardware cryptographic modules can prevent any attempt of illegal memory access.

2.3 Lack of Data Integrity Guarantee

Software implementations are lines of code written in some programming language. Since there is no mechanism to resist against the arbitrary access to the common memory space, no one can guarantee the integrity of the internal code [1], [7], [10]. Who can assure for example that an unauthorized user will not gain access to the set of instructions of the corresponding program? And if he does manage, what can eventually prevent him from altering the code, leading to a malfunction or an information leakage?

Hence, software cryptographic implementations can not guarantee the integrity of the sensitive data they are supposed to protect. Unfortunately, the possibility of tampering the software module is not the only concern. What increases the problem is the difficulty of designing tamper-evident software-based modules [5].

Indeed, software tamper detection is a difficult task for two major reasons:

I. *The software can be copied from the original host machine and modified offline, leaving no evidence of being tampered.*
II. *Assuming that the tamper is performed on the original host machine, nothing prevents the attacker from covering his tracks by simply overwriting the tampered module with the original one.*

On the other hand, hardware implementations provide a higher level of security, since the code used is being burnt on the chip [1], [10]. The hardware protection is created during the manufacturing process and includes physical barriers which prevent optical or electrical reading and any kind of alteration of the chip's contents [7].

In addition, by including extra design features during the implementation of the hardware modules, an even greater resistance to attacks can be accomplished. As an example we can mention some existing techniques that reduce the usefulness of the information enclosed in the module, by scattering the cryptographic keys and other sensitive data throughout the chip [7]. Thus, even if a successful attack is accomplished, the tamper of a single chip is not enough for the values of the key to be extracted. The cracker needs to attack multiple chips and finally to be able to piece together the information leaked by the various modules.

Preventing the illegal access to the code is not the only privilege of the hardware implementations. Tamper resistant methods, which include sensors able of detecting unusual levels of heat or light, can render the chip inoperable under an attempted attack. Moreover, these features provide the evidence that tampering has been attempted making hardware implementations tamper-evident [7].

2.4 Pseudo-Random Number Generators

At the heart of every cryptographic module there is a random number generator, whose role is to produce unpredictable numbers with the intention to form the secret key.

True random number generators cannot be implemented in software [14]. Software engineers often try to develop them by measuring physical events available in the software [13], [17]. However, this approach is quite risky because if these events are computer controlled, nothing can prevent a malicious programmer from controlling these external events and finally predicting the cryptographic key. Actually, the random number generators found in most computers are software routines implementing algorithms and are properly called pseudo-random number generators [17]. The latter cannot produce truly unpredictable numbers and constitutes a wound for most software cryptographic implementations [2], [4].

In contrast, truly random numbers can be generated in hardware with the use of some physical processes, like the thermal noise, the photoelectric or other quantum phenomena [17]. These processes, based on microscopic phenomena, are in theory completely unpredictable and thus, they are able to accumulate the necessary amount of random entropy to generate strong cryptographic keys.

2.5 Insecure Key Storage

Producing unpredictable keys is not enough to ensure their protection in case of an attack. It is essential to ensure that the key values are stored in an extremely secure place inside the memory. The arbitrary memory access of the software-based implementations sets already the key values in a great risk. However, the secure storage of the keys becomes even more crucial when it involves *long-term* or *Master keys* [1].

Long term keys in contrast to session keys, have a longer lifetime and therefore, need to be stored in a secondary memory which gives birth to additional attacks. In order to ensure their security, usually the long term keys are encrypted via other key called keys-encryption keys. Nevertheless, since the latter also need to be stored secretly, the method of the encryption key does not seem to offer any solution to the problem of the secure key storage.

In most software cryptographic modules, password-based encryption is used for the protection of the key-encryption keys [1], [2]. This method suggests the online generation of the key-encryption keys via a user-typed password. Although the password-based encryption seems to solve in a way the storage of the key-encryption keys, it has several drawbacks. First of all, it requires the human presence in order for the pass phrase to be typed [1], [14]. Therefore, this technique cannot be applicable in situations where systems or servers need to function unattained. Secondly, passwords that are user-generated have extremely low entropy which cannot eliminate the possibility of the password being guessed or exhaustively searched [1], [2], [13], [14]. Finally, due to the development of keyboard-sniffing programs, nothing can ensure the security of anything typed onto the user's computer [14].

An alternative approach for the protection of the long term keys suggests their encryption via an internal key, called *Master* key. Nevertheless, the problem of the key-storage raises again, concerning this time the master key, which in turn needs to be stored in a well "hidden" place in order to be safe from a malicious user. However, the uncontrolled memory access prohibits the secure storage of the master key making the software inappropriate for cryptographic implementations.

Hardware-based implementations seem to be the best candidate in relation to the master keys storage. The master key can be burnt on the chip during the manufacturing process rendering its extraction an extremely difficult task [1], [7], [10], [12].

2.6 No resistance to reverse engineering

One of the major fears for current cryptographic modules is reverse-engineering [1], defined as the ability to identify the function, components and dependencies of the target system. Surprisingly, reverse engineering does not threat only secret algorithm's implementation units. It can be extremely harmful even in the case of a public knowing algorithm, since it may reveal possible design flaws, which can be exploited in the future [1].

Everybody knows that there is no software resistant to reverse engineering [12]. While in software modules there is no robust memory access control, nobody can guarantee that an attacker will not figure out the system's function by reading the code instructions. Software cryptographic modules are really vulnerable to reverse engineering, because the human skills and the machines required are quite common [5]. As many systems use commercial operating systems, most crackers already have the system they want to attack at their disposal [5].

A successful reverse-engineering of a hardware cryptographic module, however, does require specific equipment and expertise [7]. In contrast to a software attack which is straight-forward and cost-effective, hardware cracking tools are much more expensive and harder to come by [11].

In addition, attackers need to be more sophisticated, with specific skills and techniques, in order to plug into a hardware device and reveal information concerning its behavior [5]. Thus, a given system implemented in tamper-resistant hardware might have a typical lifetime of 10 or 15 years before being reverse-engineered, whereas the same implementation in software might not last more than 2 or 3 years.

2.7 Vulnerability to Side Channel Attacks

Software cryptographic implementations appear to be more vulnerable to power and *timing analysis attacks* than their hardware equivalents. When a software program is executed, the contained commands are compiled to a set of assembly instructions. Usually, the power consumption of the latter follows predefined patterns which can easily be identified, even by using ordinary power analysis techniques [1]. The information collected can reveal the internal function of the cryptographic module which if properly exploited can threat the security of the cipher key.

An attacker can also employ timing analysis techniques in order to retrieve information associated to the internal state of the cryptographic or the secret key [13]. These techniques allow the attacker to extract secrets maintained in a security system by exploiting the correlation between the variations of the processing time and the operations performed [6], [11]. The only way to eliminate the threat of timing analysis attacks is to force all the computations performed to spend the same amount of time.

The difficulty of writing time-constant programs in a high level programming language can explain why software-based implementations are susceptible to the particular attack [11]. In addition, general-purposes computers cannot avoid timing variations due to various factors, such as compiler optimizations and RAM cache hits.

Although hardware-based implementations are also threatened by the above side channel attacks, special measures can be applied to mask the leakage of any information associated to power consumption and timing variations. Randomized masking methodologies and power signal filtering [1], are two of the hardware countermeasures proposed, able to reduce the effectiveness of timing and power analysis techniques and consequently, prevent the attacker from extracting values of the secret key.

3 Combining HW& SW advantages

Since software-based products fail to achieve a satisfactory level of security, why do they still remain the first choice when it comes to cryptographic implementations?

Software-based implementations are generally preferred because they are cost-effective, easy to modify or upgrade and their development requires a small design cycle [8], [9]. While the traditional (ASIC) hardware implementations offer sufficient assurance and great performances, their deterrent cost and their lack of flexibility, restrict their use to a limited number of applications [9].

The gap between software-based and ASIC implementations can be filled by reconfigurable hardware devices such as FPGAs. FPGAs are considered a highly attractive option, as they combine the flexibility of software products with a strong physical security [8], [9]. With their design requiring less time and expenses and allowing the modification and agility of the implemented algorithm, FPGAs are considered the best suited for secure cryptographic implementations.

4 Conclusions & Outlook

Software-based solutions are considered the worst candidate when dealing with the implementation of cryptographic modules, where a high level of security is demanded. The weakness of the software modules to prevent an illegal memory access, together with the threat of reverse engineering and the vulnerabilities of the OS, impose doubts on whether a software cryptographic module is able to provide the desirable data protection and integrity.

Hardware-based products are best suited for cryptographic implementations because they are able to ensure a satisfactory level of security. Providing their own physical memory, which is tightly controlled and defined, and with the use of tamper-resistant methods, they can assure the secure storage of any sort of sensitive data and eliminate the risk of a potential attack.

Despite their limitations, however, software-based modules are often developed, due to the prohibitive cost of the traditional (ASIC) hardware-based implementations. In order to exploit the advantages of each one of the two approaches, most of the current cryptographic modules rely on a FPGA implementation. FPGAs are hardware reconfigurable devices, ease to upgrade and modify, which protect physically their memory space and manage to achieve high performance in a reasonable cost.

Acknowledgment

This work is co-funded by 75% from the E.E., and 25% from the Greek Government under the framework of the Education and Initial Vocational Training Program- Archimedes.

References:

[1] Hagai Bar-El, Security Implications of Hardware vs. Software Cryptographic Modules, *Discretix Technologies*, September 30, 2002.

[2] Bruce Schneier, *Applied Cryptography*, John Wiley & Sons, 1996.

[3] Bruce Schneier, Cryptographic Design Vulnerabilities, *IEEE Computer*, Vol.31, 1998, pp. 29-33.

[4] Daniel Dariel, Choosing the Appropriate Security Technology, *M-Systems*, May 2003.

[5] Eugen Bacic, Gary Maxwell, Software Hardening & FIPS 140, *Physical Security Testing Workshop*, December 2005.

[6] Daniel J. Bernstein, *Cache-timing attacks on AES*, 2005, URL: http://cr.yp.to/antiforgery/cachetiming - 20050414.pdf.

[7] CRSS Publications, Security of Electronic Money, January 1996, URL: http://www.bis.org/publ/cpss18.pdf.

[8] K.Gaj and P.Chodowiec, Hardware performance of the AES finalists - survey and analysis of results, URL: http://ece.gmu.edu/crypto/AES_survey.pdf.

[9] T. Wollinger and C. Paar, How Secure Are FPGAs in Cryptographic Applications?, Proceedings of International Conference on Field Programmable Logic and Applications (FPL 2003), *Lecture Notes in Computer Science* Vol. 2778, 2003, pp. 91-100.

[10] Legat and JJ. Quisquater, Hardware Security for Software Privacy Support, *Electronics Letters*, Vol. 35, No. 24, Nov. 1999, pp. 2096-2098.

[11] P. Kocher, R. Lee, G. McGraw, ARaghunathan, and S. Ravi, ”Security as a New Dimension in Embedded System Design”, *ACM/IEEE Design Automation Conference*, June 2004.

[12] P.Walker,’ Hardware vs. Software Cryptography’, June 2003, *primefactors*, URL:http://www.primefactors.com/resources/index.cfm?fuseaction=article&rowid=19.

[13] P. V. Semjanov, ‘On Cryptosystems Untrustworthiness’, July 1999, URL: http://www.ssl.stu.neva.ru/psw/publications/crypto_eng.html.

[14] Jueneman Consulting, LLC, "Security Solutions for an Insecure World: Hardware vs. Software Cryptography”,URL: http://www.jueneman.com/hardware_vs_software_cryptography.html.

[15] nCipher, ‘KPMG White Paper - Response Document’,URL:http://www.ncipher.com/resources/downloads/wddfiles/KPMG_response.pdf.

[16] P.Gutmann, “An Open-source Cryptographic Coprocessor”, URL:http://www.cypherpunks.to/~peter/usenix00.pdf

[17] “Report on Review of Cryptographic Protocols and Security Techniques for Electronic Voting”,CYBERVOTE:WP2:D6/V1:2000 v1.0, URL: http://www.eucybervote.org/TUE-WP2-D6V1v1.0.pdf, January, 2002.

[18] “An Introduction to Cryptography”, URL: ftp://ftp.pgpi.org/pub/pgp/6.5/docs/english/IntroToCrypto.pdf.

[19] N. Sklavos, and O. Koufopavlou, “Mobile Communications World: Security Implementations Aspects - A State of the Art”, CSJM Journal, Institute of Mathematics and Computer Science, Vol. 11, Number 2 (32), pp. 168-187, 2003.